Introduction à Octave

Octave est un logiciel de calcul numérique, de visualisation et de programmation. C'est un logiciel libre, "clone" du logiciel commercial *MATLAB* développé par la société *The MathWorks*.

1. Octave comme "calculatrice"

1.1. Opérations élémentaires.

Voici quelques opérations:

```
octave-3.0.1:29> 3+2
ans =  5
octave-3.0.1:30> 182-23
ans =  159
octave-3.0.1:31> 45*2.3
ans =  103.50
octave-3.0.1:32> 79.2/3
ans =  26.400
octave-3.0.1:33> 2^7
ans =  128
octave-3.0.1:34> 625^(1/4)
ans =  5
```

1.2. Constantes mathématiques.

Voici quelques constantes:

```
octave:17> e
ans =  2.7183
octave:18> pi
ans =  3.1416
```

1.3. Affichage des nombres.

Voici différentes manières d'afficher des nombres:

```
octave:1> format short
octave:1> 10^6
ans =  1000000
octave:3> format short e
octave:4> e
ans =  2.7183e+00
octave:5> format short
octave:6> pi
ans =  3.1416
octave:7> format long
octave:8> pi
```

[1]Cours 3AMOS01 - 17-18 (B . Ischi)

```
ans =  3.14159265358979
octave:9> format short e
octave:10> 1602
ans =  1.6020e+03
octave:11> format long e
octave:12> 1602
ans =  1.60200000000000e+03
```

1.4. Fonctions prédéfinies.

Beaucoup de fonctions mathématiques sont prédéfinies. Voici quelques exemples:

```
octave-3.0.1:56> sin(pi/2)
ans =  1
octave-3.0.1:57> cos(0)
ans =  1
octave-3.0.1:58> sin(pi/2)
ans =  1
octave-3.0.1:59> cos(pi)
ans = -1
octave-3.0.1:60> tan(pi/4)
ans =  1.0000
octave-3.0.1:61> asin(1)
ans =  1.5708
octave-3.0.1:62> acos(-1)
ans =  3.1416
octave-3.0.1:63> atan(1)
ans =  0.78540
octave-3.0.1:64> factorial(5)
ans =  120
```

2. Les variables

2.1. Variables “numériques”.

On peut stoker des données dans la mémoire par l’intermédiaire des variables:

```
octave:1> bernard=4.67
bernard =  4.6700
octave:2> bernard
bernard =  4.6700
octave:3> 6*bernard
ans =  28.020
```

Une variable est une “case” de la mémoire dont le contenu peut changer. Pour accéder au contenu de la “case” il suffit de taper le nom de la variable.
La commande **clear** efface toutes les variables:

```
octave:4> clear
octave:5> bernard
error: ‘bernard’ undefined near line 5 column 1
```

Pour ne pas afficher le contenu d’une variable, il suffit d’ajouter un ;:

```
octave:1> bernard=4.67;
octave:2> 6*bernard
ans =  28.020
```

La commande

```
octave:2> bernard=bernard+7
```

n'est pas une équation ! (Remarquons que cette équation n'admet pas de solution !) C'est une opération d'incrémentation. Elle consiste à attribuer la valeur de bernard+7 à la variable bernard (*i.e.* à remplacer le contenu de la variable bernard par le résultat de bernard+7):

```
octave:1> bernard=3
bernard =  3
octave:2> bernard=bernard+7
bernard =  10
```

2.2. Les listes.

La commande a=[1,−3,**pi**,34] permet de stocker dans la variable a une liste de nombres.

```
octave-3.0.1:8> a=[1,-3,pi,34]
a =

    1.0000   -3.0000    3.1416   34.0000
```

La commande a(3) donne le troisième élément de la liste, c'est-à-dire π:

```
octave-3.0.1:13> a(3)
ans =  3.1416
```

Un moyen commode de générer des listes consiste à utiliser la syntaxe [3:5:27] où 3 est le premier nombre de la liste, 5 le pas et 27 la limite supérieure:

```
octave-3.0.1:3> [3:5:27]
ans =

    3    8   13   18   23
```

On peut aussi utiliser la syntaxe:

```
octave:1> a=1:2:27
a =

    1    3    5    7    9   11   13   15   17   19   21   23   25   27
```

où 2 est le pas.

La commande **length**(a) renvoie la longueur de la liste a. Par exemple:

```
octave-3.0.1:28> length([2,5,1])
ans =  3
```

Certaines opérations sont aussi définies sur les vecteurs (listes):

```
octave-3.0.1:36> a=[1,-3,pi,34];
octave-3.0.1:37> b=[2,1,4,-27];
octave-3.0.1:38> a+b
ans =

   3.0000  -2.0000   7.1416   7.0000

octave-3.0.1:39> a-b
ans =

   -1.00000   -4.00000   -0.85841   61.00000
```

La multiplication de listes de même longueur est définie comme suit:

```
>> a.*b
ans =

    2.0000   -3.0000   12.5664 -918.0000
```

La première case de a.*b contient le produit a(1)*b(1), la deuxième case de a.*b contient le produit a(2)*b(2), etc. **Attention** à la syntaxe: .* et non pas seulement * !
Les puissances sont aussi définies pour les vecteurs. Par exemple, la commande a.^2 élève chaque élément de a au carré:

```
octave-3.0.1:41> a.^2
ans =

     1.0000      9.0000      9.8696   1156.0000
```

On peut créer des listes de listes (des tableaux). Si a est un tableau:

```
a=[[1;2;3],[4;5;6]]
a =

   1   4
   2   5
   3   6
```

(ou simplement

```
>> a=[1,4;2,5;3,6]
a =

   1   4
   2   5
   3   6
```

) on accède à sa deuxième colonne ou à sa troisième ligne comme suit:

```
a(1:3,2)
ans =

   4
   5
   6

octave:4> a(3,1:2)
ans =

   3   6
```

ou simplement

```
>> a(:,2)
ans =

   4
   5
   6
>> a(3,:)
ans =
```

```
    3   6
```

Si l'argument d'une fonction prédéfinie est un vecteur a, le résultat est un vecteur dont les composantes sont les images par la fonction des éléments du vecteur a. Par exemple:

```
octave-3.0.1:70> a=[0,pi/4,pi/2];
octave-3.0.1:71> sin(a)
ans =

   0.00000   0.70711   1.00000
```

2.3. Les chaînes de caractères.

La commande a='Bonjour' permet de stocker la chaîne de caractères "Bonjour" dans la variable a. La commande a(2) affiche le deuxième caractère de la chaîne contenue dans a, c'est-à-dire o.

```
octave-3.0.1:1> a='Bonjour'
a = Bonjour
octave-3.0.1:2> a(2)
ans = o
```

La commande **findstr**(a,b) cherche la chaîne b dans la chaîne a. Elle renvoie les positions des chaînes b trouvées dans a. Par exemple:

```
octave-3.0.1:24> findstr('Bonjour, comment allez-vous ?','o')
ans =

    2    5   11   25

octave-3.0.1:25> findstr('Bonjour, comment allez-vous ?','ou')
ans =

    5   25
```

La commande **length**(a) renvoie la longueur de la chaîne a. Par exemple:

```
octave-3.0.1:26> length('Bonjour !')
ans =  9
```

2.4. Le code ASCII.

Chaque caractère est codé par un nombre selon une convention appelée le code ASCII. La commande toascii transforme un caractère en nombre et la commande char effectue la transformation inverse. Par exemple,

```
octave-3.0.1:20> toascii('a')
ans =  97
octave-3.0.1:21> toascii('Bonjour !')
ans =

    66   111   110   106   111   117   114    32    33

octave-3.0.1:22> char(97)
ans = a
octave-3.0.1:23> char([66,111,110,106,111,117,114,32,33])
ans = Bonjour !
```

2.5. Affichage des chaînes de caractères.

Pour afficher le contenu d'une variable, il suffit de taper son nom:

```
octave-3.0.1:5> a='Bonjour'
a = Bonjour
octave-3.0.1:6> a
a = Bonjour
```

Pour que le contenu d'une variable ne s'affiche pas, il suffit d'ajouter un ;

```
octave-3.0.1:7> a='Bonjour';
octave-3.0.1:8>
```

La commande **printf** permet d'afficher du texte. Par exemple:

```
octave:1> printf('Bonjour !')
Bonjour !octave:2> printf('Bonjour !\n')
Bonjour !
octave:37>
```

Le caractère \n provoque un "retour à la ligne".

3. Graphiques 2D

3.1. La commande *plot*.

Pour deux vecteurs a et b de même longueur, la commande **plot**(a,b) affiche un ensemble de points dont les abscisses sont données par a et les ordonnées pas b. Les points sont reliés entre eux. Par exemple, l'exécution des commandes

```
1 clf
2 clear
3 a=[1,0,-1,0];
4 b=[0,1,0,-1];
5 plot(a,b)
```

donne le graphique de la figure 1.

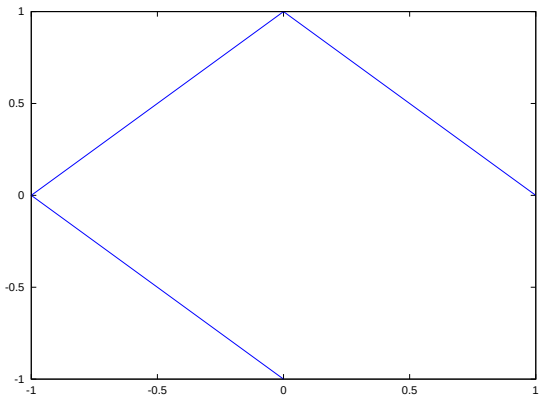


FIGURE 1. La commande **plot**

La commande **clear** efface les données contenues en mémoire et la commande **clf** efface les graphiques affichés.

La commande **plot** admet plusieurs options. Par exemple, l'exécution des commandes

```
clf
clear
a=[1,2,3,4];
b=[1,4,9,16];
plot(a,b,"@2;Points;")
title('Titre du graphique')
xlabel('texte 1')
ylabel('texte 2')
axis([0,5,-1,17],"tic[xy]")
set(gca(),'xtick', 2:2:4);
set(gca(),'ytick', -1:1:17);
set(gca(),'xticklabel', ['a=1';'b=2 ';'c';'d']);
set(gca(),'yticklabel', ['a';'b';'c';'d';'e']);
print essai.png
```

donne le graphique de la figure 2. Le @1 permet de ne pas relier les points et la commande **print** –deps essai.**eps** d'imprimer le graphique dans un fichier de type `eps`.

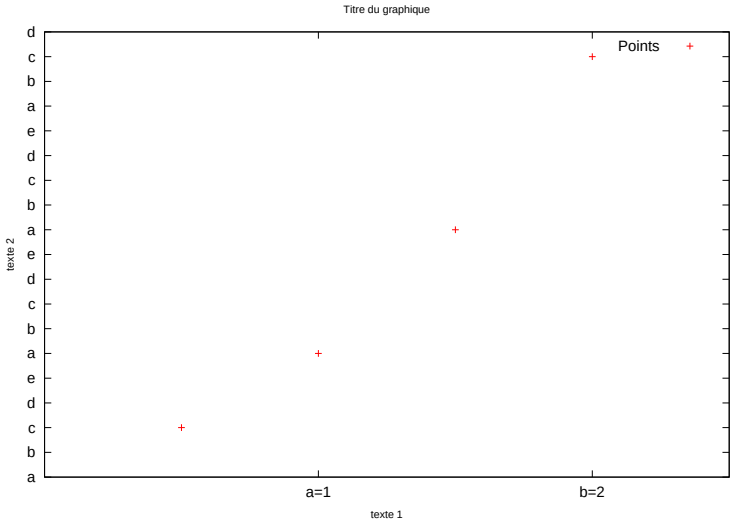
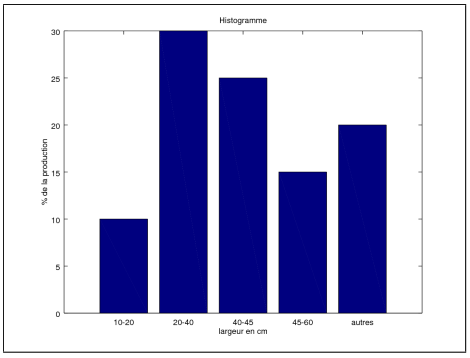


FIGURE 2. Les options de la commande **plot**

3.2. La commande *bar*.

La commande **bar** permet d'afficher des histogrammes:

```
stat=[10,30,25,15,20];
bar([1:1:5],stat)
set(gca,'xtick',[1:1:5])
set(gca,'xticklabel',{'10-20','20-40','40-45','
    45-60','autres'})
xlabel('largeur en cm')
ylabel('% de la production')
title('Histogramme')
```



4. Graphiques 3D

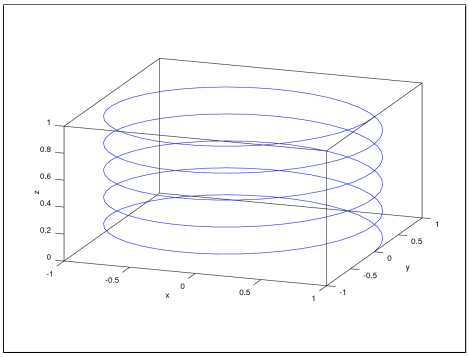
4.1. Courbes de l'espace.

Soit la fonction $\vec{r}$ de $[0, 2\pi]$ dans $\mathbb{R}^3$ définie par

$$\begin{array}{rccl} \vec{r}: & [0,2\pi] & \to & \mathbb{R}^3 \\ & t & \mapsto & \vec{r}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} \cos(5t) \\ \sin(5t) \\ \frac{t}{2\pi} \end{pmatrix} \end{array}$$

C'est l'horaire d'une mouche qui toutes les 2π secondes fait 5 tours le long d'un cercle de 1 mètre de rayon tout en montant de 1 mètre. Sa trajectoire est une courbe de l'espace: c'est une hélice de 1 mètre de rayon et dont le pas vaut $\frac{1}{5}$ de mètre, soit 20 centimètres. Le code *Octave* suivant permet de représenter cette courbe

```
t=0:(2*pi)/200:2*pi;
x=cos(5*t);
y=sin(5*t);
z=t/(2*pi);
%
plot3(x,y,z)
xlabel('x')
ylabel('y')
zlabel('z')
axis([-1,1,-1,1,0,1],"equal")
view(20,15)
```



4.2. Surfaces dans l'espace.

Soit la fonction de $\mathbb{R}^2$ dans $\mathbb{R}$ définie par $f(x, y) = x^2 + y^2$. Son graphe pour $-5 \leq x \leq 5$ et $-4 \leq y \leq 4$ est une surface de $\mathbb{R}^3$. En exécutant avec *Octave* le script donné ci-dessous, on obtient la représentation du graphe de la figure 3 (à gauche).

```
[xx,yy]=meshgrid(-5:0.5:5,-4:0.5:4);
zz=xx.^2+yy.^2;
surf(xx,yy,zz)
```

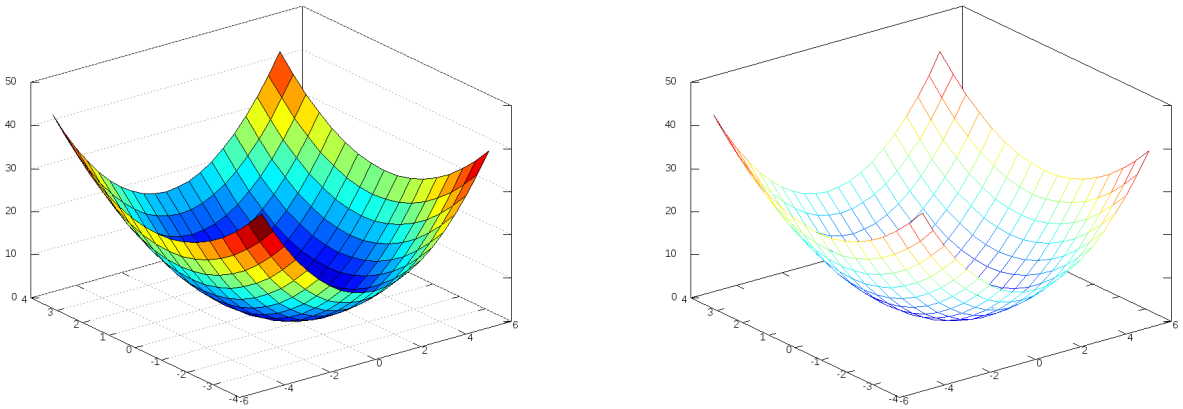


FIGURE 3. Les commandes **surf** et **mesh**

En remplaçant la commande **surf** par la commande **mesh**, on obtient la représentation à droite sur la figure 3. La commande [xx,yy]=**meshgrid**(1:1:3,1:1:2) génère deux tableaux:

```
[xx,yy]=meshgrid(1:1:3,1:1:2)
xx =

   1   2   3
   1   2   3

yy =

   1   1   1
   2   2   2
```

5. Fonctions et boucles

5.1. Les fonctions.

Il est possible de définir des fonctions avec la commande **function**. Si un fichier commence par cette commande, il doit porter le même nom que la fonction. Par exemple, pour définir la fonction

$$g(x, y, n) = x^n + y^n$$

il suffit de créer un fichier nommé `g.m` contenant les instructions:

```
1 clear
2 function z=g(x,y,n)
3   z=x^n+y^n;
4 end
```

Ainsi, on obtient

```
octave-3.0.1:3> g(1,2,3)
ans =  9
```

En effet, $1^3 + 2^3 = 9$.

5.2. Les boucles.

Octave admet deux types de boucles, **for** et **while**.
Voici un exemple de boucle **while** contenu dans un fichier nommé `essai.m`:

```
1 n=0;
2 while n<=5
3   n=n+1
4 end
```

On obtient:

```
octave-3.0.1:7> essai
n =  1
n =  2
n =  3
n =  4
n =  5
n =  6
```

Voici un exemple de boucle **for** contenu dans un fichier nommé `essai.m`:

```
1 for n=1:13:123
2   n
3 end
```

On obtient:

```
octave-3.0.1:10> essai
n =  1
n =  14
n =  27
n =  40
n =  53
n =  66
n =  79
n =  92
n =  105
n =  118
```

6. Opérations sur les vecteurs et les matrices

6.1. Opérations sur les vecteurs.

La commande **dot**(a,b) donne le produit scalaire de a et b et **cross**(a,b) leur produit vectoriel:

```
octave-3.0.1:42> a=[1,-2,3];
octave-3.0.1:43> b=[3,1,-4];
octave-3.0.1:44> dot(a,b)
ans = -11
octave-3.0.1:45> cross(a,b)
ans =

   5   13    7
```

Par conséquent,

```
octave-3.0.1:46> dot(a,cross(a,b))
ans = 0
```

6.2. Opérations sur les matrices.

La commande a=[0,1;1,0] permet de stocker une matrice dans la variable a.

```
octave-3.0.1:10> a=[0,1;1,0]
a =

   0   1
   1   0
```

La commande a(1,2) donne l'élément à la ligne 1, colonne 2:

```
octave-3.0.1:24> a(1,2)
ans =  1
```

Une matrice diagonale se définit comme suit

```
>> diag([1 3 2 5],0)
ans =
Diagonal Matrix
   1   0   0   0
   0   3   0   0
```

```
   0   0   2   0
   0   0   0   5
```

La somme de deux matrices et la multiplication par un scalaire sont données par

```
>> a=[1,2;0,1]
a =
   1   2
   0   1
>> b=[0,1;1,0]
b =
   0   1
   1   0
>> a+b
ans =
   1   3
   1   1
>> a-b
ans =
   1   1
  -1   1
>> 2*a
ans =
   2   4
   0   2
```

Le produit matriciel est donné par

```
>> a=[1,2;0,1]
a =
   1   2
   0   1
>> b=[0,1;1,0]
b =
   0   1
   1   0
>> a*b
ans =
   2   1
   1   0
>> b*a
ans =
   0   1
   1   2
```

La transposée, le déterminant et l'inverse sont donnés par

```
>> a=[1,2;0,1]
a =
   1   2
   0   1
>> transpose(a)
ans =
   1   0
   2   1
```

```
>> det(a)
ans =  1
>> inverse(a)
ans =
   1  -2
   0   1
```

Ainsi,

```
>> inverse(a)*a
ans =
   1   0
   0   1
```

et

```
>> transpose(transpose(a))-a
ans =
   0   0
   0   0
```

Le polynôme caractéristique est donné par

```
>> a=[1,2;2,1]
A =
   1   2
   2   1
>> poly(A)
ans =
   1  -2  -3
```

c'est-à-dire

$$c_A(t) = t^2 - 2t - 3$$

et en vertu du Théorème de Cayley-Hamilton, on a effectivement

```
>> A^2-2*A-3*diag([1 1],0)
ans =
   0   0
   0   0
```

c'est-à-dire

$$c_A(A) = 0$$

puisque le polynôme caractéristique d'une matrice A est un polynôme annulateur de A.
Les valeurs propres et les vecteurs propres sont donnés par

```
>> A=[1,2;2,1]
A =
   1   2
   2   1
>> [vecp,valp]=eig(A)
vecp =
  -0.70711   0.70711
   0.70711   0.70711
valp =
Diagonal Matrix
  -1   0
```

```
    0   3
```

et on peut vérifier que, effectivement,

```
>> inverse(vecp)*A*vecp
ans =
  -1   0
   0   3
```